Exploiting Narrow Values for Soft Error Tolerance

Oguz Ergin[1*], Osman Unsal[2*], Xavier Vera[3], and Antonio González[3]
[1]TOBB University of Economics and Technology, Ankara, Turkey, oergin@etu.edu.tr
[2]Barcelona Supercomputing Center, Barcelona, Spain, osman.unsal@bsc.es
[3]Intel Barcelona Research Center, Intel Labs – UPC, Barcelona, Spain, {xavier.vera, antonio.gonzalez}@intel.com

***Abstract*—Soft errors are an important challenge in contemporary microprocessors. Particle hits on the components of a processor are expected to create an increasing number of transient errors with each new microprocessor generation. In this paper we propose simple mechanisms that effectively reduce the vulnerability to soft errors in a processor. Our designs are generally motivated by the fact that many of the produced and consumed values in the processors are narrow and their upper order bits are meaningless. Soft errors caused by any particle strike to these higher order bits can be avoided by simply identifying these narrow values. Alternatively, soft errors can be detected or corrected on the narrow values by replicating the vulnerable portion of the value inside the storage space provided for the upper order bits of these operands. We offer a variety of schemes that make use of narrow values and analyze their efficiency in reducing soft error vulnerability of level-1 data cache of the processor.**

***Index Terms*—Error Correction, Soft Errors, Narrow Values, Data Cache**

I. Introduction

Alpha particles released by radioactive impurities and neutrons coming from outer space are known to cause transient errors in contemporary microprocessors [1][10]. “Single bit upsets” may arise when these particles hit intermediate capacitive nodes of processor storage components such as SRAM bitcells and latches. Since these transient errors occur due to an incorrect charge or discharge of an intermediate capacitive node, they do not cause permanent failure in the hardware and hence are termed “soft errors” in the literature. Microprocessors become more prone to soft errors with each new generation of manufacturing technology [11] and many techniques are proposed to improve soft error tolerance including redundant multithreading [12] and value duplication [7].

Architectural Vulnerability Factor (AVF) of a processor component is defined as the probability that a particle strike at any place in the component will result in an erroneous behavior in the executed program. Mukherjee et al. [9] defined architecturally correct execution (ACE) bits as the bits which are vulnerable to particle strikes. A particle hit on these ACE bits results in a visible error in the final program outcome. Similarly, a bit which does not hold any required information for architecturally correct execution and hence is not vulnerable to soft errors is defined as an unACE bit. AVF of a component is equal to the percentage of ACE bits inside the corresponding component.

Our design is generally motivated by the fact that many of the produced and consumed values in a processor are narrow where a narrow value is defined as a value that holds consecutive zeros or ones in its upper order bits [2][4][5]. These values can be represented in a simple compressed manner by just ignoring their upper order bits.

This paper proposes several techniques that leverage narrow operands to improve soft error tolerance of processors’ data-holding components. With our first technique, by identifying narrow operands and zero partitions (consecutive zeros), some portion of the narrow data becomes invulnerable to particle strikes and the total number of soft errors that affect the final program output is reduced. As a second scheme, we improve our first technique to detect and correct the particle hits that occur on the unprotected part of the narrow value by replicating the significant part of the narrow value into the storage space devoted to store the upper order bits. We further improve our technique by using storage space allocated for data partitions that hold zero values as a repository for replicated data and later we use these replicated copies of the data to detect particle hits on the stored value.

Narrow value replication was also used for soft error detection by Hu et al. in [7]. In addition to the soft error recovery scheme of [7] that replicates the narrow operands, we propose two extensions. We first show that by simply identifying narrow values it is possible to achieve a significant reduction in soft error vulnerability of the data holding components. We also show that zero holding partitions can be used as data repositories for data replication and this data replication can be combined with simple zero partition to achieve high soft error avoidance.

II. Reducing soft errors using narrow values

A. Identifying Narrow Values

Many researchers observed that a large percentage of the generated and consumed values in a processor are narrow. The narrowness of the values was previously used for performance improvement [2][4] and energy efficiency [3][4][5][13] in superscalar microprocessors. In this paper, we propose a new way of exploiting narrow values by identifying them throughout the processor for reducing soft error

Manuscript submitted: 8 May 2006. Manuscript accepted: 19 June 2006. Final manuscript received: 30 June 2006.
[*] Work done while working at Intel

vulnerability and replicating them inside the conventional storage space for error detection and correction. In order to make use of the width variations in produced and consumed values, our proposed architecture uses an additional bit called *Narrow Value Identifier Bit (NVIB)*for each data storage entry in the value holding components for identifying a stored narrow operand. This bit is set whenever a narrow value is written into the storage space. By using this bit, it is possible to identify the unneeded portion of the stored value and these bits, which are identified as "unneeded", are converted to unACE bits. Consequently, correctness of the stored narrow value is not endangered by a bit-flip caused by a particle strike if this particle strike occurs at the upper order bits. When a value is read out from the storage element, if the narrow value indicator bit is set, upper order bits are not read and the stored narrow value is sign extended to datapath width before it is ready to be used. This sign extension can be accomplished by using a simple multiplexer.

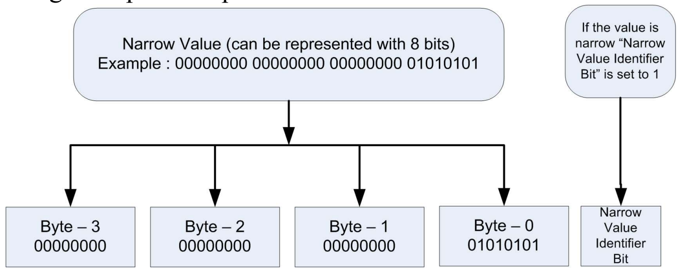
Narrow Value (can be represented with 8 bits)
Example : 00000000 00000000 00000000 01010101
If the value is narrow "Narrow Value Identifier Bit" is set to 1
Byte – 3 00000000
Byte – 2 00000000
Byte – 1 00000000
Byte – 0 01010101
Narrow Value Identifier Bit
Fig. 1 - Example of Narrow Value Identification

Fig. 1 shows an example of the narrow value identification process where a narrow value is defined as a value which can be represented with only 8 bits. The *NVIB* is checked whenever a value is read from the data storage. If *NVIB* is set, Byte–0 is simply sign extended to 32 bits and any particle strikes to Bytes 1, 2 and 3 become ineffective. There are obvious trade-offs in defining the length of a narrow operand. If a narrow operand is defined to have too few bits, then the percentage of narrow operands decreases, but the benefits of identifying the narrow values for vulnerability reduction increases since more bits are transformed into unACE bits. If a narrow operand is defined to include large number of bits, the percentage of narrow values increases but the number of protected bits in each narrow value decreases and hence the benefits also decrease. Therefore there is an optimum point for defining the number of bits in a narrow operand where the percentage of narrow operands and the number of unACE bits are optimized for best vulnerability reduction. Choice of number of bits to define the size of narrow values depends on the applications that are run.

It should be noted that the *NVIB* is itself unACE when it is indicating that a stored value is narrow. If this bit is flipped when it is indicating a narrow operand, the value is not endangered but the narrow value protection is nullified and the contents of the upper order bits become vulnerable to particle attacks. On the other hand, *NVIB* is an ACE bit when the storage space is holding a wide value since the contents of the upper order bits will be lost if a particle strike occurs on it. Therefore we call *NVIB* a "half-ACE" bit meaning that its vulnerability status depends on the contents of the value stored in the storage area.

Although *NVIB* is half-ACE, it is still partially vulnerable to particle strikes and hence increases the vulnerability of the structure it is protecting. Therefore the vulnerability reduction achieved by adding these bits must justify the slight increase in soft error vulnerability.

B. *Identifying Zero Partitions*

A variation of narrow value identification can be used to increase the chances of reducing soft error vulnerability in a processor by identifying zero-partitions instead of identifying the whole narrow values.

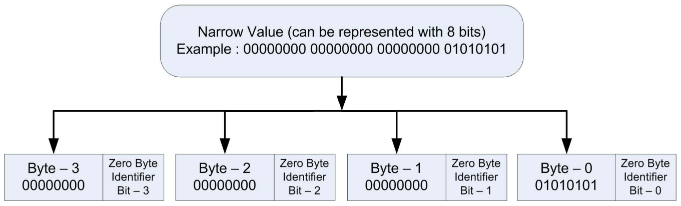
Narrow Value (can be represented with 8 bits)
Example : 00000000 00000000 00000000 01010101
Byte – 3 00000000
Zero Byte Identifier Bit – 3
Byte – 2 00000000
Zero Byte Identifier Bit – 2
Byte – 1 00000000
Zero Byte Identifier Bit – 1
Byte – 0 01010101
Zero Byte Identifier Bit – 0
Fig. 2 - Example of Zero Byte Identification

Fig. 2 shows an example of zero partition encoding process where a partition is defined to be a byte. By inserting one bit per byte, each all-zero-containing-byte can be identified and be immunized to particle strikes. When the zero byte identifier bit is found out to be set while reading the data, value is not read and instead a zero byte is provided. As it is the case with narrow value identification bits, zero partition identifier bits are also "half-ACE" since a particle strike on these bits do not jeopardize correct program execution when they indicate a zero byte. Therefore they also increase the soft error vulnerability of the component where they are added if they are not protected.

III. Replicating Narrow Values for Soft Error Detection and Recovery

Even though narrow value identification decreases the vulnerability in the data holding components of a processor, errors can still occur on the unprotected part of the narrow operand. Narrow value replication can be used for soft error detection and recovery since multiple copies of a narrow value can fit into the allocated storage space. In case of a particle hit on the entry, this particle hit can be detected by comparing stored copies with each other. Similarly, soft errors can be corrected by recovering the correct value from one of the uncorrupted copies without signaling an error or creating an exception if there are enough number of correct replicated copies.

A. *Narrow Value Replication*

In the implementation of narrow value replication, our previously proposed *NVIB* is replaced with a *Narrow Value Replicated Bit (NVRB)* which indicates that the stored value is narrow and the narrow value is replicated inside the storage space. Upon obtaining a value from the storage element, if *NVRB* of the value is set, replicated values are compared with each other for detecting or correcting a potential error. By using this bit, it is possible to detect multiple particle hits to the value or correct at least a single particle hit and recover from the error provided that there are enough copies of the value inside the storage space.

Fig. 3 shows an example of the narrow value replication process where a narrow value is defined as a value which can be represented with only 8 bits. If a value is identified as narrow, *NVRB* is set while the value is being replicated inside the storage element (4 copies of the Byte–0 are written to 32-bit storage area). *NVRB* is checked whenever a value is read from the data storage. If this bit is set, Byte–0 is simply sign extended to 32 bits after comparing all of the replicated copies with each other and making sure that all copies indicate the same value. If some of the comparisons result in mismatch, simple voting is used to decide which value to use where the highest number of identical copies inside the storage space wins.

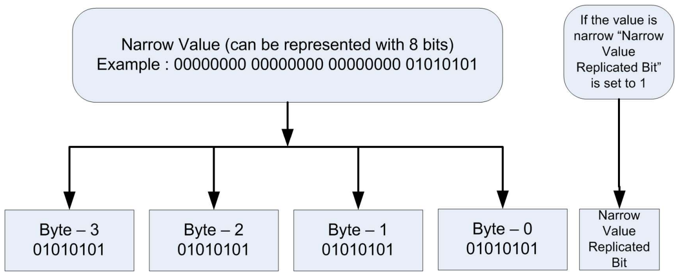
Narrow Value (can be represented with 8 bits)
Example : 00000000 00000000 00000000 01010101
If the value is narrow "Narrow Value Replicated Bit" is set to 1
Byte – 3 01010101
Byte – 2 01010101
Byte – 1 01010101
Byte – 0 01010101
Narrow Value Replicated Bit

Fig. 3. Example of Narrow Value Replication

Note that although the replicated value is protected from particle attacks, *NVRB* is itself not protected and is an ACE bit. A particle strike on this bit will endanger the correctness of the stored value at all times. Therefore unlike previously proposed *NVIB* the vulnerability increase introduced to the corresponding component by *NVRB* is not conditional. This bit can either be left unprotected to avoid increased complexity, at the expense of the soft error vulnerability increase, or it can be replicated like the narrow value. If the *NVRB* is not replicated, in some cases errors on this bit may still be recognized with additional hardware since the replicated copies will differ from each other significantly when the content of *NVRB* flips from 0 to 1.

TABLE I
ACTIONS CORRESPONDING TO SPECIFIC NUMBER OF PARTICLE STRIKES

I	J	K	L	Action
0	0	0	E_L	Corrected
0	0	E_K	E_L	If (K ≠ L) → Corrected Else → Detected
0	E_J	E_K	E_L	If (J = K = L) or (2 of [J,K,L] are equal) → Miscorrected Else → Detected
E_I	E_J	E_K	E_L	If (I = J = K = L) or (3 of [I, J, K,L] are equal) → Miscorrected Else → Detected

Table I summarizes all possible cases for different number of bit flips on the same value where I, J, K and L denote the different copies of the replicated value and $E_{I,J,K,L}$ shows the number of bit flips on the corresponding copy. As it can be seen from the table, our narrow value replication technique with a narrow value size of 8 (4 copies of the value inside), can surely correct 1 particle hit on the storage and can at worst detect 2 particle hits if the dual bit flip cannot be corrected. Although a rare event, it is also possible to have a miscorrection if most copies are hit at the same bit position and simple voting favors the faulty copies. This situation is extremely improbable and therefore we do not provide any solutions in this paper for such occasions.

In order to implement our soft error recovery mechanism, a number of comparators are needed to compare all versions of values with each other in addition to the narrow value replicated bit (and its replicated copies) per entry of the data storage element. In our example, where a narrow value is defined as 8-bits wide, there are 4 copies of the same value and six comparators are needed.

B. Replicating Data into Zero Holding Partitions

A variation of narrow value replication can be used for error detection by using the storage space allocated for zero partitions as a repository for replicating non-zero data. We propose to augment each byte inside data storage space to include a bit called "Zero-Byte, holding Replicated data" (*ZBR*). When a non-zero byte is replicated inside the storage space of a zero byte, this bit is set to indicate that the actual byte is zero and it now contains the value of another byte for error detection purposes. Similar to *NVRB*, *ZBR* is also an ACE bit at all times and same tradeoffs exist in terms of soft error vulnerability reduction and invested hardware.

Different heuristics can be applied to leverage zero portions of the values for soft error detection (and possibly recovery) in order to simplify the replication and error detection logic. The number of copies generated for a non-zero portion determines the level of protection as it is the case for narrow value replication. As the number of copies generated and the number of different places a data partition can be replicated increases, complexity of the design increases together with the level of protection obtained.

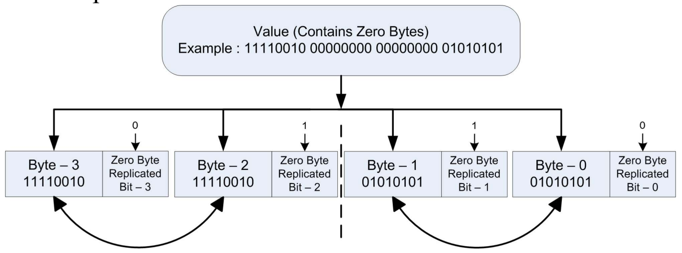
Value (Contains Zero Bytes)
Example : 11110010 00000000 00000000 01010101
0
1
1
0
Byte – 3 11110010
Zero Byte Replicated Bit – 3
Byte – 2 11110010
Zero Byte Replicated Bit – 2
Byte – 1 01010101
Zero Byte Replicated Bit – 1
Byte – 0 01010101
Zero Byte Replicated Bit – 0

Fig. 4. Example of Using Zero Bytes as Replication Repositories

As an example to demonstrate the effectiveness of our technique we present a simple heuristic to detect soft errors by taking a single copy of each byte and copying each byte only in one (or optionally two) different places. Fig.4 shows an example of the heuristic where the 32-bit value is divided into two parts and within each part each byte is allowed to be replicated to the other byte. Replicating bytes only within 16-bit partitions simplifies the design since when the *ZBR* bit of a byte is set it is known that the replicated byte is the adjacent byte within the 16-bit grouping. In Fig. 4, value to be stored contains 2 zero bytes (Byte–1 and Byte–2). Using our heuristic, Byte–0 is replicated to Byte–1 and Byte–3 is replicated to Byte–2. When the value is accessed from the storage space, the error detection logic will detect that Byte–1 and Byte–2 holds replicated data and will compare Byte–3 with Byte–2 and Byte–1 with Byte–0. If there is a mismatch, processor will signal the detected error. If an error is not detected, contents of Byte–2 and Byte–1 will be discarded and replaced with zero-bytes.

An optimization is possible to extend the protection to cover the cases when Byte–3 and Byte–2 are zero-bytes and Byte–1 and Byte–0 hold valid data or vice versa (this case is similar to 16-bit narrow value replication with two uppermost bytes are zero). Our heuristic can be extended to detect this case and lower order 2 bytes can be copied to the upper order 2 bytes. When the error detection logic detects that both ZBR bits within a 16-bit group are set, it understands that the stored bytes are copied from the other 16-bit group and compares the corresponding bytes with each other for error detection (Byte–0 with Byte–2 and Byte–1 with Byte–3).

Although the optimized heuristic of replicating data into zero bytes has larger soft error detection coverage, a better alternative would be to combine zero byte identification for vulnerability reduction and data replication for detection. When both of the bytes in a byte pair are zero, both *ZBR* bits can be set and their zero byte status can be identified without any data replication inside them. Upon reading the data, whenever error detection logic detects that both bits are set, it just replaces any data stored inside the bytes with zeros. This way zero bytes are protected from particle attacks while non-zero bytes can be replicated into a zero byte for error detection if the byte next to them is zero.

IV. Results and Discussions

In order to evaluate the proposed schemes we used a cycle-accurate simulator that simulates a microarchitecture that resembles Intel's Pentium 4 [6]. Table II shows the vulnerability reduction achieved in the level-1 data cache on average across all SPEC2k benchmarks for various schemes proposed in this paper. As the numbers reveal, decrease in soft error vulnerability increases as the number of bits in a narrow operand decreases and at 4-bits there is an optimum point that returns the best soft error coverage.

TABLE II
Soft error reduction achieved in level-1 data cache

Decrease in soft error vulnerability for different narrow value widths	*32 bit*	27.1%
	16 bit	38.0%
	8 bit	45.7%
	4 bit	47.1%
	2 bit	45.7%
Replicating narrow values	***Correction (10 bits)***	45.0%
	Detection (16 bits)	49.6%
Identifying and replicating zero partitions	***Simple replication***	22.1%
	Extended replication	30.6%
	Simple replication with zero identification	62.4%

By replicating narrow values inside the provided storage space 45.0% of the errors can be corrected and around 49.6% of the errors can be detected. The numbers show that for different levels of complexity we can achieve high percentage of error coverage by exploiting narrow values.

It should be noted that for individual benchmarks percentage of narrow values can vary and the optimum point for the best error tolerance can be different. Therefore the benefits of exploiting narrow values highly depend on the processor component and the workload.

V. Conclusion

We proposed soft error avoidance, detection and recovery mechanisms which protect the stored values by either identifying narrow values or replicating parts of the operands into the already available storage space. None of our schemes result in IPC degradation. Exploiting narrow values for error tolerance turned out to be a cost effective solution although it provides protection only when the stored value is narrow. Our first technique of just identifying narrow values with a single bit results in a soft error vulnerability reduction of as high as 47.1% in the level-1 data cache on average for SPEC2k. In order to protect the vulnerable part of the narrow value we proposed replicating these values into the provided storage space which lead to 49.6% error detection or 45.0% error correction for single bit upsets. Our last technique which replicates non-zero data partitions inside the storage space of zero partitions can detect 22.1% and 30.6% of the single bit errors with our simple and extended schemes respectively. Our hybrid scheme that combines replication and zero identification together avoids 40.3% and detects 22.1% of the errors which sums up to a total of 62.4% error reduction.

References

[1] Baumann, R., "Soft Errors in Advanced Computer Systems", in *IEEE Design & Test of Computers*, 2005

[2] Brooks, D., and Martonosi, M., "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance", in *HPCA*, 1999

[3] Canal R., et al., "Very Low Power Pipelines using Significance Compression", in *MICRO*, 2000

[4] Ergin, O., et al., "Register Packing: Exploiting Narrow-Width Operands for Reducing Register File Pressure", in *MICRO*, 2004

[5] Gonzalez, R., et al. "A Content Aware Register File Organization", in *ISCA*, 2004

[6] Hinton, G., et al., "The Microarchitecture of the Pentium 4 Processor", *Intel Technology Journal*, Q1, 2001

[7] Hu, J., et al., "In-Register Duplication: Exploiting Narrow-Width Value for Improving Register File Reliability", in DSN 2006

[8] Memik, G, et al., "Increasing Register File Immunity to Transient Errors", in *DATE'05*, 2005

[9] Mukherjee, S. S., et al., "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor", in *MICRO*, 2003

[10] Phelan, R., "Addressing Soft Errors in ARM Core-based Designs", White Paper, ARM, December 2003

[11] Semiconductors Industry Association (SIA), International Technology Roadmap for Semiconductors 2003, http://public.itrs.net/Files/2003ITRS/Home2003.htm.

[12] Vijaykumar, T. N., et al., "Transient-Fault Recovery Using Simultaneous Multithreading", in *ISCA*, 2002

[13] Villa, L., et al., "Dynamic Zero Compression for Cache Energy Reduction", in *MICRO*, 2000